PraaS: Verifiable Proofs of Property as-a-Service with Intel SGX

Istemi Ekin Akkus
Nokia Bell Labs

Ivica Rimac
Nokia Bell Labs

Ruichuan Chen
Nokia Bell Labs

***Abstract*—Industry 4.0 presents an immense opportunity to create novel automation solutions. These solutions will be largely augmented using analytics based on Machine Learning (ML), requiring diverse and often large datasets. Obtaining such datasets is not always possible for individual entities. Although data marketplaces can facilitate collaborations, the lack of trust relations among these entities can pose a significant barrier on the extent of these interactions.**

In this paper, we advocate that Trusted Execution Environments (TEEs), such as Intel SGX, can provide an effective solution to this problem. We present PraaS (Proof-as-a-Service), a system enabling dataset owners to obtain *verifiable proofs* that their datasets satisfy certain properties, without exposing their confidential datasets. These proofs can be checked by third parties to obtain assurances about the datasets they plan to use in their applications *before actual usage*. PraaS achieves these properties with readily available TEEs that incur low monetary cost, and can support large static and streaming datasets. For example, PraaS can produce a uniform sample from a static dataset of 5M hashes in 267 seconds, and compute statistics for streaming batches of integers with a rate of 200K/sec in less than 820 ms, along with their verifiable proofs. PraaS also allows easy and on-demand customizability for various property computations in C/C++ and Python with enclave templates and automated configuration.

1. Introduction

In Industry 4.0, many organizations will augment their automation solutions using with ML analytics. These analytics approaches will require diverse and large-scale datasets [2], [22], [23], [40], which may not always be possible for individual organizations to possess, requiring them to collaborate with others. Data marketplaces can facilitate such collaborations [5], [11], [12], [15], [32], [38]; however, the lack of trust relations among these entities can pose a significant barrier on the scale and extent of their data exchanges.

This trust requirement can be seen from three perspectives. First, the datasets typically contain confidential information that is important for the sharing entities (i.e., dataset owners). The risks of sharing such datasets may be too large in terms of operational security, financial information and logistics. Second, the entities utilizing the shared datasets (i.e., dataset users) most likely base their ML analytics solutions on these datasets, which in turn power their automation solutions for critical infrastructure. As a result, the quality, validity, integrity and properties of such datasets play an important role for their use of these datasets. Finally, the content of the datasets plays an increasing role from a privacy perspective, especially in the cases where the datasets contain individuals' data. This increasing importance is already reflected in laws and regulations, such as GDPR in Europe [14], HIPAA and CCPA in the US [10], [20]. These regulations will certainly reflect on the owners and users of such private data in terms of compliance, either as an enforced requirement (e.g., by regulatory bodies) or self-imposed public relations act (e.g., to be seen as 'good' by general public).

Besides the trust requirements, any solution to this problem should also consider *utility*. One cannot always determine a dataset's utility *before* it is shared and used, because it often depends on the user's application and/or the type of the dataset. As a result, customizability and on-demand computation of properties become requirements, imposing additional constraints on the solution. Coupled with a dataset's type (i.e., static or streaming), its size and the rate of new data, these constraints force one to consider practical issues, such as cost, scalability with size and latency of proof generation. These considerations lead to trade-offs between these issues and trust assumptions, prompting the question: What is the minimum cost and trust to obtain the maximum performance?

We advocate that Trusted Execution Environments (TEEs) [3], [29] can provide an effective solution to this problem: TEEs are readily available on major cloud providers, and cost as low as 0.09 Euro/hour [6], [8]. They are performant enough to handle large datasets and streaming data (Section 7). Finally, they offer 'acceptable trust', where the root of trust is the TEE hardware instead of the cloud provider. As a result, TEEs can simultaneously address the trust requirements among various entities: confidentiality/privacy constraints of dataset owners and utility/customizability demands of dataset users.

In this paper, we present **PraaS** (**Pr**oof-**a**s-**a**-**S**ervice), a system that enables dataset owners to obtain verifiable proofs for properties of their confidential datasets. To use TEEs in this fashion, we propose a generic extension to TEE computations: When the TEE initializes, it generates an *ephemeral* public/private keypair and embeds the public key in the attestation quote. When the property computation is finished, the TEE signs the output with its ephemeral private key. The attestation quote with the public key and the signed TEE output comprise the *verifiable proof*: a third party can check the validity of the quote, and then the signature on the output using the public key in the quote to ensure the output was indeed produced faithfully in the TEE.

In the next section, we present motivational use cases, followed by background and related work on how to achieve verifiable proofs for property computations (Section 3). We list our assumptions and goals in Section 4. We

then describe the design, implementation and evaluation of **PraaS** (Sections 5, 6 and 7). We conclude with a discussion of our approach's implications and future work in Section 8. The code of **PraaS** is publicly available [36].

2. Motivational Scenarios

Federated Learning offers a potential solution for the privacy requirement: The model is sent to dataset owners, who compute their updates over their private data, and send the updates back to the model owner [1], [33], [34], [42]. TEEs can also enable collaborative learning with dataset privacy. Ohrimenko et al. [35] propose that dataset owners agree on a model and training code, deploy them in an SGX enclave in the cloud, attest to it and upload their datasets. Citadel [43] and Chiron [21] use TEEs to not only preserve dataset privacy but also the confidentiality of the model coming from a different collaborator. Here, local and global model updates are applied with verifiable and trusted code that does not expose private data.

In these cases, participants never expose their datasets to other participants, preserving the confidentiality and privacy. However, this approach creates a potential *fairness problem*: Malicious participants may not contribute useful data to the training; yet they can still benefit from the trained model. Such participants cannot be easily detected because the usefulness of the dataset requires one to see and compute over the data, which conflicts with the confidentiality/privacy requirement. Similarly, before a dataset user obtains a dataset, the user may want to know whether it will satisfy the requirements of a specific use case. Consequently, dataset owners need to convince potential collaborators that their datasets can provide value, which may not be easy: besides the confidentiality/privacy requirement, giving the dataset away would also contradict with the goal of monetizing it. In addition, the data may not be fully available yet (e.g., streaming data).

3. Background & Related Work

3.1. Background: SGX Remote Attestation

One popular commercially available TEE solution is Intel's Secure Guard Extensions (SGX). SGX allows user- and OS-level code to define private memory regions (i.e., enclaves), whose contents are protected for confidentiality and integrity. With remote attestation [27], users obtain an *attestation quote*, ensuring them that they are interacting with a genuine SGX enclave on an up-to-date system with the latest trusted computing base (TCB). In the quote, of particular interest are the cryptographic measurement and report data fields (i.e., `MRENCLAVE`, `REPORTDATA`). `MRENCLAVE` value refers to the code and configuration running inside the enclave. `REPORTDATA` value refers to the enclave data at the initialization and can be used for creating a secure channel with the enclave.

There are two remote attestation flavours: **Enhanced Privacy ID (EPID) based attestation** requires the platform and TEE user contact the online attestation service operated by Intel [25]. It is deprecated due to Intel's focus on cloud [27], [28], [30]. **Elliptic Curve Digital Signature Algorithm (ECDSA) based attestation** is enabled via the Intel SGX Data Center Attestation Primitives (DCAP), and allows cloud operators to build their own attestation services. This approach is available on Xeon processors that also provide larger Enclave Page Caches [27]. Cloud applications benefit from ECDSA-based attestation, because the interactions between Intel and the attestation service are transparent to the applications.

3.2. Related Work

Zero-Knowledge Proofs (ZKPs) [17] can be used to prove the integrity and correctness of a property computation without revealing private data. However, despite recent advances to improve their performance [19] and usability [9], the performance of ZKPs with large data and arbitrary computations remains low, making them unsuitable for scalability and customizability.

NIZK [41] proposes creating non-interactive ZKPs within a TEE. They incorporate the Lua interpreter in an enclave, such that users supply a script with public and private input. The computation happens inside the TEE and the result is embedded in the attestation quote. Although their usage of attestation quotes as proof is similar, there are several crucial differences that make NIZK unsuitable for our purposes. First, NIZK sets the `REPORTDATA` field in the attestation quote *after* the computation. In a streaming scenario, this approach would require *one attestation per batch*, making it unsuitable for continous computations for streaming data. In contrast, **PraaS** requires *one attestation per stream* (Section 7). Second, it is unclear how a private dataset can be shared with the TEE *before* the attestation; the script's private input is in the context of the ZKP (i.e., not exposed as part of the result in the attestation quote).

Genés-Durán et al. [16] define a data exchange protocol with a free sample (DEFS) and integrate it into a data marketplace [15]. The main goal is to provide a dataset sample to potential buyers in a decentralized data exchange. As such, it only works for static datasets, because the sampling protocol requires encrypted dataset items to be uploaded beforehand (i.e., no streaming). Additionally, it only considers sampling and not the generation of verifiable proofs for custom property computations.

4. Assumptions & Goals

In this section, we first present the actors in **PraaS**. We then present our assumptions about them and the threat model we consider, followed by our goals.

4.1. Actors

Dataset owner: A dataset owner wants to prove that the dataset satisfies certain properties and thus, is of value to dataset users. For example, the dataset owner may want to monetize a dataset by making it available on a marketplace [5], [11], [12], [38]. Similarly, the owner may want to convince potential partners and collaborators that a dataset will be beneficial for them [4], [43].
Dataset user: A dataset user may want to obtain a dataset and use it for an application scenario and ML analytics solution. The dataset can be bought on a marketplace, but before purchasing, the dataset user would like to obtain

confidence about the properties of the dataset and its potential utility. Similarly, a participant in a collaboration may want more confidence that others are contributing fairly and are not trying to free-ride from others' datasets.
PraaS provider: The **PraaS** provider operates the necessary infrastructure to produce proofs of property for dataset users. We envision **PraaS** to be run in a cloud setting, so that this entity could be an actual infrastructure provider that is offering **PraaS** on TEEs, or a separate entity that is provisioning TEE resources from an infrastructure provider and operating **PraaS**.

4.2. Threat Model & Assumptions

We assume that the **PraaS** and the infrastructure provider do not collude with the dataset owner and dataset user. Neither dataset owners nor dataset users need to trust the infrastructure provider, except for the assumptions that standard security practices are in-place (e.g., physical security of infrastructure) and the TEE infrastructure is kept up-to-date (e.g, TEE firmware updates are performed regularly). We think that these assumptions are reasonable in today's world, where many companies trust the cloud infrastructure providers even without TEEs.

There are several ways to share the proofs of property of a dataset with potential users (e.g., private communications, announcements, marketplace). These methods may also include the dataset owner demonstrating ownership of a dataset and the proofs of property computed over it (e.g., by signing any created proofs with the owner's private key). For brevity, we do not describe these protocols, and leave any issues regarding them outside our scope.

Many TEE manufacturers, including Intel, take a meticulous approach regarding vulnerabilities and their patches [31]. Nonetheless, we consider attacks on the underlying TEE hardware outside our scope.

4.3. Goals

Ideally, a dataset owner would produce verifiable proofs of the properties of the dataset, and send the proofs to potential users, who would then verify them.
Customizability: Some properties may be generic and can apply to many use cases. For example, checking the formatting requirements of a dataset or ensuring that the data values are feasible may be common among many applications. Similarly, informational properties of a dataset may be useful for many applications (e.g., percentiles, standard deviation). However, many properties may not easily be forecast or enumerated, because a dataset's utility often depends on the use case and the properties of the dataset.
Practicality: Many datasets are large, requiring considerations about performance, cost and trust. Processing large datasets in a timely manner will certainly affect their usefulness. Furthermore, some datasets may be continuously generated (i.e., streaming data), requiring a performant solution to handle them. Similarly, the cost plays a role, because generating a proof for a dataset should be less expensive than the monetization goal. These practical aspects lead one to make trade-offs that may also include the trust assumptions while producing such proofs.

With these aspects in mind, our goal is to design a practical and general system that supports dataset owners' and users' needs, regarding properties, dataset sizes and dataset types. The system should be performant enough to support large-scale static and streaming datasets, incur low cost and have acceptable trust assumptions.

Algorithm 1 Enclave initialization, ephemeral keypair generation and output signing

```
1:  procedure INITENCLAVE()
2:      keypair ← generateRSAKeypair()
3:      report ← getSGXAttestationReport()
4:      report.REPORTDATA ← hash(keypair.publicKey)
5:      return report, keypair.publicKey
6:  procedure DOCOMPUTE(ENCRYPTEDDATA)
7:      data ← decryptWithPrivateKey(encryptedData)
8:      output ← computeProperty(data)
9:      signature ← signWithPrivateKey(output)
10:     return output, signature
11: procedure HANDLEREQUEST()
12:     report, enclavePublicKey ← InitEnclave()
13:     quote ← getSGXQuote(report)
14:     sendToUser(quote, enclavePublicKey)
15:     encryptedData ← receiveFromUser()
16:     output, signature ← DoCompute(encryptedData)
17:     sendToUser(output, signature)
```

5. Design

In this section, we first introduce the technical building block to extend the trust in a TEE to third parties that were not involved with the TEE creation and computation. We then present property computation functions (PCFs) that extract the desired property from a confidential dataset. We also describe a dataset owner's workflow to generate a proof of property for a confidential dataset, and how potential dataset users can verify that proof. We instantiate **PraaS** with Intel SGX because of its availability.

5.1. Enclave-signed Output

One challenge of using SGX enclaves for generating proofs of properties that are verifiable by third parties (i.e., dataset users) is due to the fact that their original goal was to give guarantees and confidence to the user that launched the enclave but not others: after establishing a session with it, a user would be confident that the computation result was obtained inside that enclave. Convincing other users, however, requires more: even if the user supplies the attestation quote to the others, there is still an incentive not to be honest when sharing the computation result (e.g., claiming a dataset satisfy a property to monetize it). This problem exists because the attestation quote and the enclave computation result are not linked together.

To address this shortcoming, we propose a simple and generic extension that can be applied to any enclave computation. Here, the enclave code produces an ephemeral public/private keypair during initialization, and embeds the secure hash of the public key in the attestation quote (i.e., `REPORTDATA` field). When the enclave finishes its computation, its signs the result with the ephemeral private key of the enclave. Consequently, a third-party user can link the quote and the computation result, by verifying the

quote, extracting the public key and checking the signature on the output.

Linking Quote and Enclave Output. Algorithm 1 shows the pseudocode of our extension. When the **PraaS** server receives a user request, it creates an enclave and requests the enclave's attestation report (Line 12 in Algorithm 1). In the procedure for enclave initialization, an ephemeral public/private keypair is generated first (Line 2), and then the attestation report is extracted via an SGX SDK call (Line 3). Before returning the report, the procedure stores the secure hash of the ephemeral public key to the `REPORTDATA` field of the attestation report. Note that lines 2-4 happen inside the enclave; thus, the private key is only accessible inside the enclave.

Subsequently, the platform attests to the report and signs it (Line 13) [24]. The server then sends the attestation quote (i.e., signed report) and the enclave's public key to the requesting user (Line 14). Any party with the quote can then extract from the attestation quote the hash of the ephemeral public key of the enclave and compare it with the hash of the received public key.

The user then performs the regular remote attestation checks [27] about the validity of the quote with the Attestation Service (e.g., Intel, Microsot Azure Attestation). If the quote is valid, the user encrypts the confidential data and supplies it to the enclave (Line 15), triggering the enclave to first decrypt the encrypted data (Line 7) and then to compute the property of the data (Line 8). Afterwards, the enclave signs the computation result with its ephemeral private key to produce a signed enclave output (Line 9). The server then returns the signed output to the user (Line 17).

Third-party Verifiable Proof. The user now has two pieces: the remote attestation quote obtained at the initialization of the enclave and the signed enclave output. Recall that the remote attestation quote includes information about the enclave's ephemeral public key. These two pieces constitute the *verifiable proof* that the computation result was produced inside a secure enclave: A third party user can first verify the attestation quote that states the integrity and correctness of the TEE hardware, and then use the embedded public key to verify the signature on the enclave output.

The enclave output contains the secure hash of the input to the enclave as well as the computation result. Note that the input and computation result can be complex with use of high-level data structures (e.g., JSON-encoded strings correctly interpreted in the enclave code).

This approach has two advantages: First, the proof verification can be performed even after the enclave is terminated, not requiring any coordination among the TEE user and the third party (e.g., dataset owner and user). Second, the attestation is decoupled from the computation, so that arbitrarily complex and long computations (e.g., computations with large datasets) as well as computations with periodical output (e.g., computations of streaming data) can be supported (Section 7), unlike producing the report after the computation [41].

5.2. Property Computation Functions

A property computation function (PCF) is the logic to extract a desired property from a dataset. Examples of such desired properties can include checks about formatting, compatibility to a schema as well as internal consistency and feasibility checks. For example, the 'age in years' column in a health dataset could be an integer and should be a positive value less than 100. Similarly, the 'BMI' values should correspond to the 'height in cm' and 'weight in kg' values. PCFs can also include computation of statistical properties of a column (e.g., percentiles, mean) as well as arbitrarily complex and customized application logic that can gauge the utility of the dataset for a specific use case.

We envision that these functions can comprise a catalogue, from which a dataset owner or a potential dataset user can pick. The owner can also create a custom PCF if the property will show the dataset's value to potential users. Similarly, the dataset owner may also accept custom PCFs from potential users, so that they can better determine whether the dataset will benefit their applications. Accepted custom PCFs could then become part of the catalogue for future use.

PCF Correctness. The PCF developer may not necessarily be the same entity as the dataset owner or the dataset user. However, the PCF code is available to both the dataset owner and potential dataset users, so that they can check its correctness and its intentions. For dataset owners, this property is important because they need to check whether the PCF is trying to blatantly leak confidential data. For example, a malicious PCF may try copying private data to the output, an attacker may combine multiple PCF outputs to leak private data, or establish covert channels. When the dataset owner cannot confidently determine such attempts, the owner has the prerogative to decline any PCFs in order not to risk such attacks, use differential privacy techniques [13], or allow only a certain number of PCFs on a given dataset. The inspection of such PCFs can also be performed by auditors or trusted third parties on behalf of the dataset owners.

On the other hand, dataset users need to check whether the PCF is indeed computing the property of the dataset they are interested in. Otherwise, a malicious dataset owner may try to supply a dataset that, on the best case, would be useless for the user, and on the worst case, can create vulnerabilities and attacks on the user's application (e.g., poisoning the ML model). Note that a potential dataset user also has the right not to accept a proof with a given dataset, if the PCF result indicates that the data is not going to be beneficial for the user's application.

Enclave Templates. Writing a custom PCF from scratch could be difficult, considering that most of the application enclaves are written in low-level languages like C/C++ [7], [26]. To facilitate the easy creation of a PCF, we provide a template that covers the general steps of an enclave, including the generation of an ephemeral public/private keypair, retrieval of an attestation quote with the public key, decryption of encrypted private data and signing of the computation result. The user only supplies a single C/C++ class with a pre-defined function that is called from the template code during execution. The server then compiles the template and the user-supplied code into an enclave binary. Note that the user can check that this binary is correct by compiling it locally and comparing it with the measurement value in the attestation quote.

To support PCFs written in high-level languages such

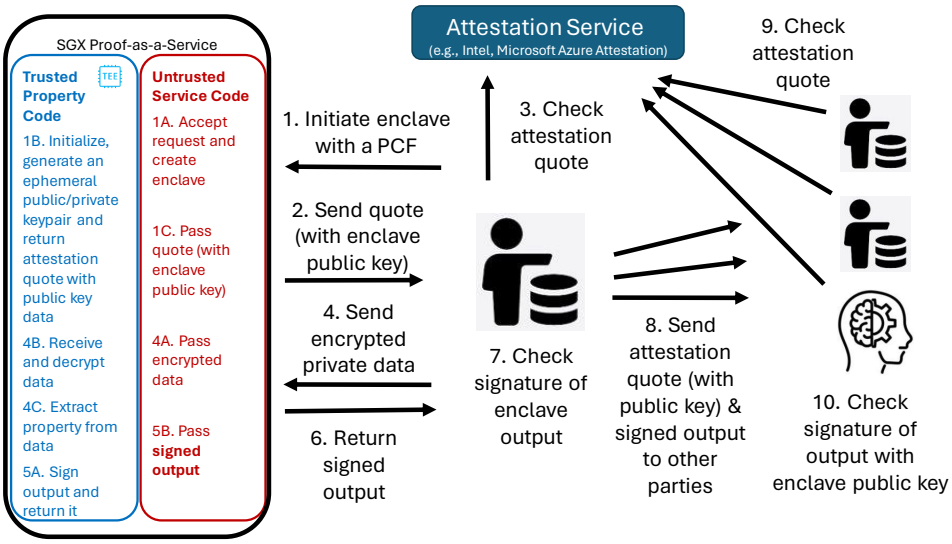

Figure 1. High-level PraaS workflow.

as Python, we allow users to upload a script for the property computation and use a libOS [37], [39]. Inside **PraaS**, the general steps are packaged into an enclave using the libOS and then instantiated, while the user-supplied script becomes part of the enclave input besides the private dataset. Here, the attestation quote only reflects the generic enclave code; however, the enclave output can also include the secure hash of the input script. The generic enclave code performing this action is also available to the dataset owners and potential dataset users, who can check its correctness. Both templates, build scripts for PCF enclaves and **PraaS** server code are publicly available [36].

PCF Integrity. In C/C++ enclaves, the supplied template and the PCF code are compiled into the enclave binary, with which the enclave is launched. As a result, the measurement (i.e., `MRENCLAVE`) in the attestation quote covers the PCF code. Because the PCF code, the enclave template and the build scripts are available to dataset owners and dataset users, they can reproduce this value by compiling the enclave binary themselves.

In Python enclaves, we utilize a libOS [37], [39], and the `MRENCLAVE` value only reflects the template code. However, the enclave template computes the hash of the PCF code and includes it in its signed output. The dataset owners and dataset users have access to the template and PCF code, so that they can reproduce both the `MRENCLAVE` value of the template and the PCF code's hash. After loading the PCF code but before executing it over the confidential dataset, the dataset owner can retrieve these hash values from the template code executing in the enclave. Alternatively, the template code can check the PCF's expected hash value supplied by the dataset owner and stop in case of a mismatch.

As an alternative to the above approach, one could also include the PCF code while creating the build artifact (e.g., a container), so that it is covered by the `MRCENCLAVE` value. One disadvantage of this approach is that the build artifact and the measurement value would be specialized for each individual PCF. Considering that the libOS causes to increase the trusted computing base, we opted on fixing the enclave template code with the libOS, so that its code can be inspected, measured and reused for different PCFs. A minor disadvantage is that the integrity of the PCF supplied as an input needs to be computed separately and compared with the value in the enclave-signed output.

5.3. Proof Generation

Figure 1 depicts the high-level workflow of **PraaS** using Intel SGX, in which a dataset owner produces a proof of property using **PraaS** and sends it to potential users. Here, the **untrusted service code** facilitates the initialization and termination of the TEE as well as proxying any (encrypted) messages between the user's client software and the TEE. The **trusted property code** runs inside the TEE and performs the general operations described in Section 5.2 as well as the property computation.

The dataset owner first requests a TEE enclave, either with a PCF from the catalogue or by uploading a custom PCF (Step 1 in Figure 1). The untrusted service code initializes the requested enclave (Step 1A). During its initialization, the enclave generates an ephemeral public/private keypair as well as a cryptographic report of its status that includes the hardware status (i.e., genuine SGX on an up-to-date system with latest TCB) and the secure hash of the enclave code (i.e., hash of the enclave binary compiled from the enclave template and the PCF). Any user can match this value with a locally calculated value to ensure that it is what they expected. The secure hash of the enclave's ephemeral public key is included in the attestation quote's `REPORTDATA` field [24] (Step 1B), allowing any user to link the attestation quote to the enclave's computation result via the enclave's signature. Recall that these steps are in the enclave template, allowing users to check their correctness. The attestation quote and the enclave's ephemeral public key are then returned to the client via the service code (Step 1C).

After receiving the attestation quote and the enclave's public key (Step 2), the client checks the quote's validity via the Attestation Service (Step 3), either with Intel for EPID-based attestation [25] or a provider (e.g., Microsoft Azure Attestation) for ECDSA-based attestation. If the

attestation succeeds (i.e., genuine and up-to-date enclave with correct code), the session continues.

The dataset owner then encrypts the private dataset with an ephemeral session key, and the session key with the enclave's public key. The owner sends the encrypted datasetand key to the enclave, and triggers the PCF (Step 4). The untrusted service passes the encrypted dataset to the TEE (Step 4A). The enclave code decrypts the symmetric key with its private key and decrypts the private dataset with the symmetric key (Step 4B). The enclave code then computes the property from the dataset and produces the computation result. The result can contain additional metadata about the dataset if needed (Step 4C). When finished, the enclave produces the signed enclave output (Step 5A) and sends it to the dataset owner via the service code (Step 5B & 6). It contains:

(i) the hash of the (decrypted) input dataset (H_{ID}),
(ii) the property computation result (PCR),
(iii) the hash of the PCR (H_{PCR}),
(iv) the hash of the PCF (H_{PCF}) (for Python) and
(v) a signature (Sig) over the above fields with the enclave's ephemeral private key generated at initialization.

5.4. Proof Verification

With the public key obtained in Step 2, the dataset owner then checks the signature by computing the secure hashes of the dataset (i.e., to match H_{ID}) and the computation result (i.e., to match H_{PCR}) (Step 7). Checking H_{ID} ensures that the untrusted service code did not tamper with the encrypted data exchange.

Afterwards, the dataset owner announces or sends to potential dataset users the availability of the verifiable proof of property about the dataset (e.g., data marketplace) (Step 8). The proof consists of the attestation quote with the enclave's ephemeral public key received in Step 2 and the signed enclave output received in Step 6.

A potential dataset user then verifies the proof of property: The user first checks the validity of the attestation quote with the Attestation Service (Step 9). Then the signature on the enclave output is checked with the public key from the attestation quote (Step 10). The successful checks indicate that the property of the dataset claimed by the owner was correctly computed in a TEE. Finally, the user determines if the property of the dataset satisfies the user's requirements. If so, the user contacts the owner to obtain the dataset (e.g., via the data marketplace).

6. Implementation

Table 1 shows our implementation details with Intel SGX. Our first implementation, based on Microsoft Azure's sample code [7], supports property computation functions (PCFs) written in C/C++. It consists of a server, the enclave template, and a catalogue of PCFs. The enclave template defines the external calls as well as the common functions about enclave initialization, attestation quote generation and output signing.

The PCFs in our catalogue were compiled into binaries with the enclave template (Table 2). *Sampling* produces a uniform sample from the dataset. *Non-repetition+sampling* first ensures that no items were repeated before sampling. *Statistics* computes statistical properties (e.g., mean, percentiles) of the items. *Sampling+statistics* first obtains a sample and then computes the statistics on the sampled items. Each example is written in less than 225 lines of C/C++ code, making the expansion of the catalogue easy. The Python client can trigger an enclave, with a PCF from the catalogue or with a custom PCF, perform attestation with the Attestation Service (i.e., Microsoft Azure Attestation) and communicate with the enclave as a static or streaming data source.

Our second implementation is in Python and supports PCFs in Python. The server accepts a script and its dependencies from the client, and launches it inside an SGX enclave with gramine libOS [18], [39]. After attestation, the client sends the private data. We created a Python PCF with 45 lines of code that counts the data items in training datasets (i.e., cifar10, mnist, fashionmnist, spamnet).

TABLE 1. **PRAAS** IMPLEMENTATION DETAILS IN LINES OF CODE. THE PYTHON ENCLAVES ARE CREATED WITH GRAMINE LIBOS.

Enclaves in	Untrusted Service	Enclave Template	Client (Python)	Example PCFs
C/C++	860	1080 (+jsonlib)	489	100-225
Python	583	N/A	369	45

TABLE 2. EXAMPLE C/C++ ENCLAVES.

Example Enclave	Supported Data Type	Enclave Size	# of pages
Sampling	`string[]`	5.5MB	H: 256K, S: 8K
Non-rep.+sampling	`string[]`	5.5MB	H: 256K, S: 8K
Statistics	`int[]`	6.1MB	H: 1K, S: 1K
Sampling+statistics	`int[]`	6.1MB	H: 1K, S: 1K

7. Evaluation

In this section, we evaluate the feasibility of **PraaS**for use with static and streaming datasets with two example PCFs. We performed all measurements on an SGX-capable VM on Azure (Standard DC2sv3: 2vCPUs, 16GB RAM) using Microsoft Azure Attestation. The server and client were on the same VM to facilitate easier communication and reduce any network issues.

For **static datasets**, we use the sampling PCF for strings as a representative example. Sampling with strings and its proof can be a powerful primitive, even if the dataset contains other types of data (e.g., images, video): The dataset owner first creates a list of hashes belonging to the data items. These hashes are used to obtain a sample and proof. Afterwards, the dataset owner prepares the actual sample with those data items, whose hashes are in the proof. A dataset user then checks whether the hashes in the proof match the hashes of the sampled items. For our experiments, we create static datasets with up to 5M hashes (with base16-encoded SHA256).

For **streaming data**, we imagine there is a broker distributing data coming in batches to subscribers. The broker and subscribers are interested in reducing bandwidth usage, such that the broker filters some batches and

TABLE 3. CLIENT-SIDE MEAN (STDEV) LATENCIES FOR ENCLAVE INITIALIZATION WITH A PCF (AT LEAST 20 RUNS).

	setup_enclave (ms)	verify_quote (ms)
Sampling	1794.5 (49.8)	175.2 (6.4)
Non-rep.+sampling	1844.0 (30.3)	174.5 (3.4)
Statistics	424.3 (125.8)	183.4 (7.0)
Sampling+statistics	442.9 (140.6)	184.0 (29.4)

TABLE 4. CLIENT-SIDE MEAN LATENCIES FOR HASH DATASETS WITH 'SAMPLING' ENCLAVE (AT LEAST 20 RUNS).

	1M	2M	3M	4M	5M
encrypt_data (s)	6.2	12.4	18.7	24.9	31.2
send_wait_result (s)	47.1	94.0	141.3	188.3	235.5
verify_signature (s)	0.07	0.14	0.2	0.27	0.35
Total (s)	53.4	106.7	160.3	213.6	267.1

does not send them if certain properties are not satisfied. However, the subscribers would like to ensure that the broker is filtering correctly. The broker uses **PraaS** to compute the property in question and to produce a proof. Accordingly, it only sends the proof but not the actual batch of data. We vary the batch rate up to 200K/sec.

7.1. Proof Generation & Verification Latency

We show **PraaS**'s latencies for generating and verifying a proof of property, measured by the client to better reflect a dataset owner's perspective. We also report on the client's verification of the quote and the signature. In addition, we break down the latency of the general steps in the enclave: initialization and quote generation, decryption of private data, PCF computation and signing the output.

Enclave setup. Table 3 shows the latencies of operations related to the initialization of a PCF in an enclave. At the server, setting up an enclave involves the initialization of the enclave and obtaining an attestation quote with an ephemeral public key from it. The initialization depends on the enclave type due to its memory configuration (i.e., stack and heap) as well as its PCF code, and causes most of the latency. In contrast, obtaining the attestation quote takes about 60 ms for all types. Similarly, the quote verification is about the same across different enclaves (and also dataset sizes), which is expected.

Static datasets. Table 4 shows the client latencies for creating a sample with its proof. Here, the dominating factor is the communication of the encrypted data and waiting for the PCF result. When investigated at the server-side, we find that the property computation takes a much smaller time compared with the reception and decryption of private data. For example, for 5M hashes, the property computation and signing on average take about 140 ms and 90 ms, respectively.

Streaming data. Table 5 shows the client-side latency values for handling a batch that includes data encryption, transmission, proof generation and signature check for *that current batch*. **PraaS** can handle up to 200K/sec under 820 ms. Note that the enclave is reused for multiple batches, so the quote verification happens only once.

TABLE 5. CLIENT-SIDE MEAN LATENCIES FOR STREAMING DATA WITH 'STATISTICS' ENCLAVE (100 BATCHES, AT LEAST 20 RUNS).

Rate per sec	10K	20K	50K	100K	150K	200K
Mean (ms)	82.6	115.33	211.4	411.3	619.8	812.9
Stdev (ms)	3.27	3.06	1.99	4.08	5.31	7.44

7.2. Proof Size

Recall that the proof consists of the attestation quote with the enclave's public key and the signed enclave output. Using the same format as Azure's sample code [7], the attestation quote and public key are about 11.10KB with base16-encoded hashes. The size of the signed enclave output depends on the PCF's result (PCR). The fixed fields (i.e., H_{ID}, H_{PCR} and Sig) are 640 bytes in total.

8. Discussion & Future Work

We advocated that readily available TEEs can provide a cost-effective and performant solution in Industry 4.0 settings, where lack of trust relations among entities can hinder collaborations. To address this problem, we presented **PraaS**, a system enabling dataset owners to obtain verifiable proofs of properties for their confidential datasets to convince potential dataset users about the value of the datasets. We showed that **PraaS** can handle large static and streaming datasets with low cost, high scalability and low latency.

One question is whether trusting the TEE manufacturer is an acceptable trust assumption. We note that similar assumptions are being already made: In Machine-Learning-as-a-Service, confidential datasets are exposed to the provider. Similarly, cloud office software (e.g., Google Docs, Microsoft Office) can see confidential data. **PraaS** relaxes these assumptions to only that the TEE hardware is genuine and up-to-date, so that neither the service nor the infrastructure provider has access to the confidential data and can tamper with the property computation.

Detection of covert channels during the inspection of the PCF code can be especially difficult. If the potential user colludes with the **PraaS** or infrastructure provider, it may be possible to leak confidential data. Separating the roles of these entities and using PCFs from trusted sources can reduce such risks for dataset owners. The dataset owners also have the prerogative to decline any custom PCF code from potential users if they are not confident about its behavior after inspection.

One potential issue is that a potential dataset user is only interested in the property computation output as given in the proof. To prevent leakage of sensitive data, the dataset owner can utilize differential privacy (DP) [13] in the output. **PraaS** can facilitate easy usage of DP by integrating it into the enclave template. One can also combine ZKPs with **PraaS**, especially with sampling: the sample would be much smaller than the original dataset making it practical to produce ZKPs about properties, and **PraaS** would provide the proof that the sample was produced uniformly and correctly. Privacy concerns about leaking sensitive data with the sample can also be addressed this way. We leave these features for future work.

Acknowledgements

We thank our anonymous reviewers and our shepherd for their feedback and suggestions to improve this paper. We also thank our colleagues in our lab for their feedback on initial versions of **PraaS**.

References

[1] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

[2] Angelos Angelopoulos, Emmanouel T Michailidis, Nikolaos Nomikos, Panagiotis Trakadas, Antonis Hatziefremidis, Stamatis Voliotis, and Theodore Zahariadis. Tackling faults in the industry 4.0 era—a survey of machine-learning solutions and key aspects. *Sensors*, 20(1):109, 2019.

[3] TrustZone for Cortex-A - Arm. https://www.arm.com/technologies/trustzone-for-cortex-a. Last accessed on 04.03.2024.

[4] Data Collaboration Service - AWS Clean Rooms - AWS. https://aws.amazon.com/clean-rooms/. Last accessed on 04.03.2024.

[5] Data Marketplace - AWS Data Exchange - AWS. https://aws.amazon.com/data-exchange/. Last accessed on 04.03.2024.

[6] EC2 On-Demand Instance Pricing - Amazon Web Services. https://aws.amazon.com/ec2/pricing/on-demand/. Last accessed on 04.03.2024.

[7] Sample Code for Intel SGX Attestation using Microsoft Azure Attestation. https://github.com/Azure-Samples/microsoft-azure-attestation/tree/master.

[8] Pricing - Linux Virtual Machines — Microsoft Azure. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#pricing. Last accessed on 04.03.2024.

[9] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[10] California Consumer Privacy Act — State of California - Department of Justice - Office of the Attorney General. https://www.oag.ca.gov/privacy/ccpa. Last accessed on 04.03.2024.

[11] Databricks Marketplace — Databricks. https://www.databricks.com/product/marketplace.

[12] Datarade — Find the right data, effortlessly. https://datarade.ai/.

[13] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[14] What is GDPR, the EU's new data protection law? - GDPR.eu. https://gdpr.eu/what-is-gdpr/. Last accessed on 04.03.2024.

[15] Rafael Genés-Durán, Oscar Esparza, Juan Hernández-Serrano, Fernando Román-García, Miquel Soriano, Achille Zappa, Martin Serrano, Susanne Stahnke, Birthe Böhm, Edgar Fries, et al. Data marketplaces with a free sampling service. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 333–338. IEEE, 2022.

[16] Rafael Genés-Durán, Juan Hernández-Serrano, Oscar Esparza, Marta Bellés-Muñoz, and José Luis Muñoz-Tapia. Defs—data exchange with free sample protocol. *Electronics*, 10(12):1455, 2021.

[17] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[18] GitHub - gramineproject/gramine: A library OS for Linux multi-process applications, with Intel SGX support. https://github.com/gramineproject/gramine. Last accessed on 04.03.2024.

[19] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, pages 339–358. Springer, 2006.

[20] Summary of the HIPAA Privacy Rule — HHS.gov. https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html. Last accessed on 04.03.2024.

[21] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[22] Capturing the true value of Industry 4.0. https://www.mckinsey.com/capabilities/operations/our-insights/capturing-the-true-value-of-industry-four-point-zero. Last accessed on 04.03.2024.

[23] How AI, ML and Industry 4.0 affect plant automation. https://www.plantengineering.com/articles/how-ai-ml-and-industry-4-0-affect-plant-automation/. Last accessed on 04.03.2024.

[24] Attestation Service for Intel Software Guard Extensions (Intel SGX): API Documentation. https://www.intel.com/content/dam/develop/public/us/en/documents/sgx-attestation-api-spec.pdf. Last accessed on 04.03.2024.

[25] Intel SGX: Intel EPID Provisioning and Attestation Services. https://www.intel.com/content/www/us/en/content-details/671370/intel-sgx-intel-epid-provisioning-and-attestation-services.html. Last accessed on 04.03.2024.

[26] Code Sample: Intel Software Guard Extensions Remote Attestation End-to-End Example. https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html. Last accessed on 04.03.2024.

[27] Attestation Services for Intel Software Guard Extensions. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html. Last accessed on 04.03.2024.

[28] Rising to the Challenge - Data Security with Intel Confidential Computing - Intel Community. https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141. Last accessed on 04.03.2024.

[29] Intel Software Guard Extensions (Intel SGX). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html. Last accessed on 04.03.2024.

[30] New Intel chips won't play Blu-ray disks due to SGX deprecation. https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/. Last accessed on 04.03.2024.

[31] Affected Processors: Transient Execution Attacks & Related Security... https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/processors-affected-consolidated-product-cpu-model.html. Last accessed on 04.03.2024.

[32] IOTA Data Marketplace. https://blog.iota.org/iota-data-marketplace-cb6be463ac7f/.

[33] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

[34] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[35] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious {Multi-Party} machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, 2016.

[36] Nokia-Bell-Labs/proof-as-a-service. https://github.com/Nokia-Bell-Labs/proof-as-a-service. Last accessed on 06.05.2024.

[37] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 955–970, 2020.

[38] Snowflake Data Marketplace — Snowflake Data Cloud. https://www.snowflake.com/en/data-cloud/marketplace/. Last accessed on 04.03.2024.

[39] Chia-Che Tsai, Donald E Porter, and Mona Vij. {Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658, 2017.

[40] Amit Kumar Tyagi, Terrance Frederick Fernandez, Shashvi Mishra, and Shabnam Kumari. Intelligent automation systems at the core of industry 4.0. In *International conference on intelligent systems design and applications*, pages 1–18. Springer, 2020.

[41] Bingsheng Zhang, Yuan Chen, Jiaqi Li, Yajin Zhou, Phuc Thai, Hong-Sheng Zhou, and Kui Ren. Succinct scriptable nizk via trusted hardware. In *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26*, pages 430–451. Springer, 2021.

[42] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[43] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. Citadel: Protecting data privacy and model confidentiality for collaborative learning. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 546–561, 2021.